MALNAD COLEGE OF ENGINEERING

Under the auspices of M.T.E.S

(An autonomous institution under Visvesvaraya Technological University, Belgaum)
Hassan – 573201, Karnataka, India

Report on Course Title: Full Stack Development

“Mini freelance hub”

Submitted by team number :8

Name	USN
Tushar Sunil Havaldar	4MC23IS119
Santosh Basavraj Shetty	4MC23IS097
Kiran B R	4MC24IS401
Vedanth S G	4MC23IS123
Subhash B R	4MC23IS109

Submitted to

Mr. Krishna Swaroop A

(Assistant Professor Dept. of ISE)

Department of Information Science & Engineering

Malnad College of Engineering

Hassan– 573202

2025-26

Date of Submission:

Table of Contents

1.Abstract

In the modern digital era, freelancing has emerged as a major source of employment, allowing individuals to work independently without being associated with a full-time company or organization. With the increasing demand for digital services, clients frequently look for skilled professionals to complete short-term tasks such as graphic design, content writing, web development, video editing, and social media promotions. However, many existing freelancing platforms are complex, highly competitive, and charge high commission fees, making it difficult for students, beginners, and small-scale freelancers to enter the freelance market. To overcome these challenges, this project proposes the development of a **Mini Freelancing Platform** using Django, which is easy to use, cost-effective, and beginner-friendly.

The Mini Freelancing Platform acts as a bridge between **clients** who require services and **freelancers** who possess relevant skills. The platform allows users to register either as clients or freelancers. Clients can post projects with details such as project title, description, budget, and deadline. Freelancers can browse the listed projects and place bids with their proposed price, delivery time, and message explaining why they are suitable for the job. The client can review all bids and select the most appropriate freelancer based on their experience, bid amount, proposal, and rating. After project completion, clients can provide reviews and ratings for freelancers, helping future clients make better hiring decisions.

The system is developed using Django, which provides a secure, scalable, and modular structure for web application development. The database used is SQLite (or optional PostgreSQL), which stores essential data such as users, projects, bids, reviews, and roles. The platform promotes **self-employment, digital earning opportunities, and skill development**, especially for students and fresh graduates. This project demonstrates how technology can empower individuals by providing them with real-time income-generating opportunities through online freelance work. Overall, the Mini Freelancing Platform simplifies online hiring and earning processes, ensuring a smooth, secure, and efficient freelancing experience for both clients and freelancers.

2.Introduction

Freelancing has transformed the way people work in today’s digital world. It allows individuals to earn money using their skills from anywhere, without the need for a traditional full-time job. The rise of the internet has created thousands of online work opportunities in various fields such as website development, graphic designing, digital marketing, writing, translation, and data entry. Students, homemakers, fresh graduates, and professionals are now turning to freelancing as a flexible way to earn income and gain experience. At the same time, businesses and individuals prefer hiring freelancers for short-term work instead of hiring permanent employees, as it is more cost-effective and efficient.

Even though freelancing platforms like Fiverr, Upwork, and Freelancer.com exist, they are often difficult for beginners to use. They charge high commissions, have strict approval processes, and involve complex user interfaces. Many beginners struggle to get their first project or showcase their talent due to high competition. There is a need for a **simple, user-friendly, and low-cost freelancing platform** that is suitable for small projects, easy communication, and beginner-level users. This need led to the development of the Mini Freelancing Platform.

The **Mini Freelancing Platform**, developed using Django, serves as a digital marketplace where clients and freelancers can interact, collaborate, and work together. Freelancers can create accounts, showcase their skills, and bid on available projects. Clients can post their work requirements, evaluate bids, and hire freelancers based on their proposals. The system also allows both parties to communicate and share project requirements easily. After the work is completed, clients can provide feedback and ratings to help freelancers build their profile and credibility.

The main goal of developing this Mini Freelancing Platform is to **promote skill-based earning opportunities**, especially among students and fresh graduates who seek practical experience and income while learning. It encourages **self-employment, entrepreneurship, and digital work culture**, reducing the dependency on traditional jobs. The platform is designed to be **simple, secure, and effective**, making it suitable for small-scale freelancing needs.

3.Objectives

The primary objective of the Mini Freelancing Platform is to develop a simple, efficient, and user-friendly system that connects clients with freelancers for small-scale digital tasks. The platform aims to eliminate the barriers faced by individuals who want to start freelancing but lack access to large, complex websites. By providing a clean and easy-to-navigate interface, the platform enables clients to post projects effortlessly and freelancers to explore opportunities without confusion. One major objective is to support beginners—such as students, new freelancers, and individuals with limited technical knowledge—by giving them a space to showcase their skills and gain work experience.

Another key objective of the system is to enable clients to receive multiple bids on the projects they post. This ensures that clients have the freedom to choose the freelancer who best fits their budget, timeline, and skill requirements. For freelancers, the platform aims to offer equal opportunities by allowing them to submit competitive bids, present their expertise, and manage their work more professionally. In this way, the system encourages healthy competition, improves service quality, and enhances user satisfaction.

The platform also aims to streamline communication between clients and freelancers. Clear communication is essential for successful project completion, and the system aims to reduce misunderstandings by providing a structured workflow—from project posting to bid submission, selection, delivery, and final approval. Additionally, the system aims to increase transparency in the selection process, allowing clients to review freelancer profiles, previous work, and proposal details before making a decision.

A further objective is to maintain an organized and efficient project management flow. Once a bid is accepted, both parties should be able to track the status of the project, view deadlines, and exchange information easily. The platform also aims to store all data securely, manage user roles effectively, and support smooth navigation throughout the system.

Overall, the broader objective of this Mini Freelancing Platform is to create a digital space where clients can access affordable services and freelancers can find work opportunities without any barriers.

4.Requirements

4.1 Software Requirements

The Mini Freelancing Platform requires a set of software tools and technologies to ensure smooth development and operation. The system is built using **Python**, and **Django** serves as the primary framework for backend development. For development purposes, the application can run on any modern operating system such as **Windows 10/11**, **Ubuntu Linux**, or **macOS**. The default database used during development is **SQLite**, as it is lightweight and easy to configure. However, for production-level deployment, more powerful database systems like **PostgreSQL** or **MySQL** may be used to handle a larger number of users and transactions. A modern code editor such as **Visual Studio Code** or **PyCharm** is required for writing and managing the project code. Developers also use **Git** for version control and **Postman** optionally for API testing if the project is expanded with REST APIs. The project additionally requires essential Python libraries such as Django, Pillow (for image handling), and other dependencies that support authentication, database management, and file uploads. A modern web browser such as **Google Chrome** or **Microsoft Edge** is needed to access and test the application interface during development.

4.2 Hardware Requirements

The hardware requirements for developing and running the Mini Freelancing Platform are minimal, as Django applications do not demand high-end resources. A computer with at least a **dual-core processor** and **4 GB of RAM** is sufficient for basic development, although **8 GB RAM** is recommended for smoother multitasking and faster processing. Storage needs are modest; around **10 to 20 GB of free disk space** is adequate to install Python, Django, the database, and development tools. Using an **SSD drive** is recommended because it significantly improves application loading time, file handling, and overall system performance. A display with a minimum resolution of **1280×720** ensures that code editors and browsers render properly during development. Since installation of packages, testing, and documentation often require internet access, a stable network connection is also essential.

5. System Design

The system design of the Mini Freelancing Platform is based on a **three-tier architecture**, which consists of the **Presentation Layer (Front-end/UI)**, **Application Layer (Django Backend)**, and **Data Layer (Database)**. This structure ensures that the system is modular, scalable, and easy to maintain. The platform is developed using the Django framework, which follows the **MTV (Model–Template–View)** pattern, similar to MVC, and separates business logic, data handling, and user interface clearly.

At the **Presentation Layer**, users interact with the system through a web browser. The main user types are **Clients** and **Freelancers**. Clients can register, log in, create and manage projects, view bids, and select freelancers. Freelancers can register, log in, view available projects, place bids, and track their applied projects. The user interface is built using HTML, CSS, and basic JavaScript (or Bootstrap if needed) to provide a clean and simple experience. Each page, such as login, dashboard, project listing, project details, and bid submission, is rendered using Django templates.

The **Application Layer** contains the main logic of the system and is implemented using Django views, models, and forms. This layer is divided into three main modules (apps):

1. **Users** – Handles user registration, login, and profile management. It defines a User model with basic fields like name, email, password, and role (client or freelancer). This app is responsible for authentication and role-based behavior.
2. **Projects** – Manages all operations related to projects posted by clients. It defines a Project model with attributes such as title, description, budget, client reference, and creation date. Views in this app handle creating projects, listing all projects, and showing details of a single project.
3. **Bids** – Manages bids placed by freelancers on client projects. It defines a Bid model that links a freelancer to a specific project along with bid amount, message, and date. This app handles adding bids and viewing all bids for a particular project.

When a **client posts a new project**, the request flows from the browser to the Django view in the Projects app, which validates the form data and saves it in the database using the Project model. When a **freelancer views projects**, the system fetches the project list from the database and renders it on the UI. When a **freelancer submits a bid**, the Bids app processes the form, creates a new Bid entry linked to the selected project and freelancer, and stores it. The client can then open a project’s details page, where all related bids are retrieved from the Bids table and displayed. Once the client selects a freelancer, the project status can be updated accordingly.

The **Data Layer** consists of a relational database (by default, SQLite in Django). It stores tables for Users, Projects, and Bids. Relationships are defined using foreign keys, such as each Project linked to a Client (User with role “client”), and each Bid linked to both a Freelancer (User with role “freelancer”) and a Project. This structured data design ensures that operations like “get all bids for a project” or “get all projects by a client” are efficient and straightforward.

Security and access control are also part of the system design. Django’s built-in authentication system is used to manage sessions and passwords securely (hashed). Only logged-in users can create projects or place bids. Simple role-based logic is applied so that only clients can create projects and only freelancers can place bids, ensuring proper usage of the platform.

Overall, the system design focuses on **simplicity, modularity, and clarity**. By separating Users, Projects, and Bids into different apps and using Django’s MTV pattern, the Mini Freelancing Platform becomes easy to extend in the future. Features like reviews, messaging, or payment integration can be added later without changing the core structure.

6. Database design

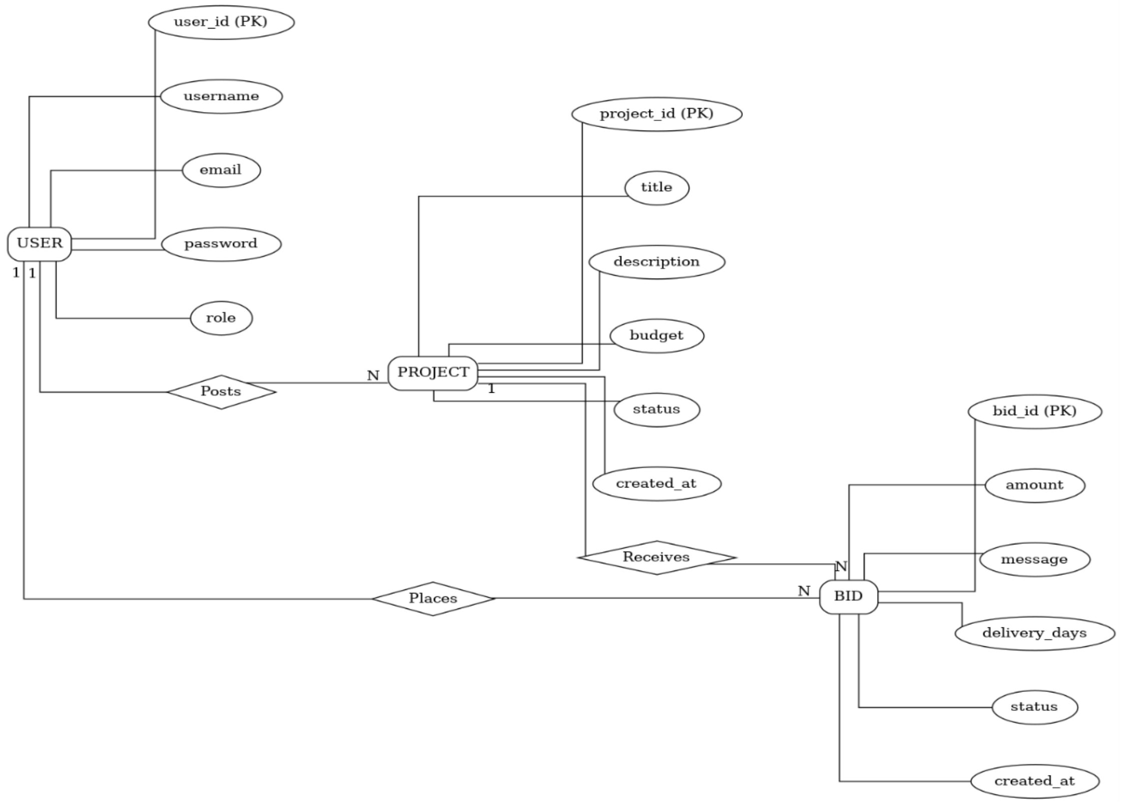
Below is a simple, clear database design for your Mini Freelance project (three apps: **users**, **projects**, **bids**). I include an ER summary, table schemas (columns + types + constraints), SQL CREATE TABLE examples (Postgres style), and short Django models.py snippets so you can use whichever fits your workflow.

6.1 ER summary (entities & relationships)

- **User** — stores both clients and freelancers (role distinguishes).
- **Project** — posted by a client; one client → many projects.
- **Bid** — submitted by a freelancer for a Project; one project → many bids, one freelancer → many bids.
- **Optional**: Service (if you want predefined service types/categories) and Review (rating after completion).

Relationships:

- Project.client_id → User.id (1:N)
- Bid.project_id → Project.id (1:N)
- Bid.freelancer_id → User.id (1:N)



6.1.1 ER diagram

6.2 Tables (columns, types, constraints)

1) users

- id — SERIAL / bigserial, PRIMARY KEY
- username — VARCHAR(150), UNIQUE, NOT NULL
- email — VARCHAR(255), UNIQUE, NOT NULL
- password_hash — VARCHAR(255), NOT NULL
- role — VARCHAR(20), NOT NULL ('client' or 'freelancer')
- full_name — VARCHAR(200), NULLABLE
- bio — TEXT, NULLABLE
- profile_pic — VARCHAR(255), NULLABLE (path/URL)
- date_joined — TIMESTAMP WITH TIME ZONE, DEFAULT now()

Notes: keep authentication fields compatible with Django's auth if using custom user model.

2) projects

- id — SERIAL, PRIMARY KEY
- client_id — INTEGER, FOREIGN KEY → users(id), NOT NULL
- title — VARCHAR(255), NOT NULL
- description — TEXT, NOT NULL
- budget — NUMERIC(10,2), NULLABLE (or NOT NULL)
- status — VARCHAR(20), NOT NULL DEFAULT 'open' — (open, assigned, in_progress, submitted, completed, cancelled)
- assigned_bid_id — INTEGER, NULLABLE, FOREIGN KEY → bids(id) (when client selects a bid)
- created_at — TIMESTAMP WITH TIME ZONE, DEFAULT now()
- updated_at — TIMESTAMP WITH TIME ZONE, DEFAULT now()

3) bids

- id — SERIAL, PRIMARY KEY
- project_id — INTEGER, FOREIGN KEY → projects(id), NOT NULL
- freelancer_id — INTEGER, FOREIGN KEY → users(id), NOT NULL
- amount — NUMERIC(10,2), NOT NULL
- message — TEXT, NULLABLE
- delivery_days — INTEGER, NULLABLE
- status — VARCHAR(20), DEFAULT 'pending' (pending, accepted, rejected)
- created_at — TIMESTAMP WITH TIME ZONE, DEFAULT now()

Constraint: optionally add UNIQUE (project_id, freelancer_id) if a freelancer can bid only once per project.

6.3 Indexes & Performance Tips

To ensure efficient performance of the Mini Freelancing Platform, proper indexing is essential. Foreign key fields such as projects.client_id and bids.project_id should have indexes to speed up queries, especially when retrieving all projects for a specific client or all bids for a specific project. Although many databases automatically index foreign keys, it is important to verify this depending on the database system being used. If the platform supports project searching by title, implementing a **full-text index**—such as PostgreSQL's GIN index—can significantly improve search speed and accuracy. As the user base grows, scaling becomes important, and moving from SQLite to PostgreSQL is recommended for better performance and concurrency handling. Additional enhancements like using Redis for caching frequently accessed data and implementing pagination for long lists can further improve system responsiveness.

6.4 Normalization & Data Integrity

The database design of the Mini Freelancing Platform follows **Third Normal Form (3NF)** to maintain clean and well-structured data. By separating core entities such as users, projects, and bids into their own tables, redundancy is reduced and data consistency is improved. Foreign key constraints help maintain referential integrity, ensuring that every project is linked to a valid client and every bid is linked to an existing freelancer and project. The use of ON DELETE CASCADE rules guarantees that when a project is deleted, its associated bids are automatically removed, preventing orphaned records and maintaining a clean database state. This design not only keeps the data logically organized but also supports reliable long-term maintenance.

6.5 Workflow & Transaction Safety

To preserve system accuracy and prevent inconsistent data states, certain actions—especially critical ones like accepting a bid—must be handled using database transactions. During the workflow of assigning a freelancer to a project, the system first begins a transaction, then updates the selected bid's status to *accepted* and marks all other bids for the same project as *rejected*. The project record is simultaneously updated by assigning the accepted bid and changing its status to *assigned*.

7. Implementation

The Mini Freelancing Platform is implemented using Django's Model–View–Template (MVT) architecture, but instead of dividing the system into multiple apps, the entire functionality is developed inside **one single Django app**. This approach keeps the structure simple, easy to manage, and suitable for student projects or small-scale freelance platforms. The single app contains all models, views, templates, URL configurations, forms, and logic required to run the system. Users, Projects, and Bids are implemented as separate models within the same app, which reduces complexity and avoids cross-app referencing. This unified structure also makes development faster because all files are located in one module, and developers do not need to maintain separate urls.py or views.py for each component.

7.1 Model Overview

The **User model** inside the single app handles all users who interact with the system. Instead of having different tables for clients and freelancers, a single consolidated model stores all user information. A role field is used to distinguish between **client** and **freelancer**, making the system more flexible. Each user record stores details such as full name, email, password, role, and an optional bio. Clients are responsible for posting projects, whereas freelancers use the same system to browse available projects and place bids on tasks they are interested in. By using one single user model with role-based logic in the views, the system remains clean, simple, and easy to maintain, while still supporting all role-specific features that a freelancing platform requires.

The **Project model** is also implemented within the same app and represents the jobs posted by clients. Each project stores important information such as the client who posted it, project title, description, and budget. The system records when each project was created and may include an optional status field indicating whether a project is open, assigned, or completed. Because all models are within the same app, linking the project to the user who posted it through a foreign key becomes straightforward. The project structure supports a one-to-many relationship: a single client can post many projects, but each project belongs to exactly one client. This model acts as the core of the platform because it serves as the marketplace where freelancers find work opportunities.

The **Bid model** connects freelancers to projects inside the same app. A bid consists of the freelancer's offer, quoted amount, and a short proposal message explaining how they intend to complete the work. Each bid record includes references to both the project and the freelancer using foreign keys. Since all models live in the same app, the relationships between them are direct and easy to manage. A single project can have multiple bids, and a freelancer can place bids on many different projects. This model captures the competitive bidding process that occurs on freelancing platforms and allows clients to compare offers before selecting the best freelancer.

7.2 URL Routing

In this implementation, the platform uses a **single urls.py file inside the app** and a minimal main urls.py in the project directory. The main URL file includes one path that directs all routes to the app's URL configuration. Inside the app-level urls.py, routes are organized for handling user registration, login, project creation, project listing, bid placement, and dashboard views. This makes the codebase extremely easy to navigate. Because everything is located inside one app, routes such as /register/, /login/, /projects/, /projects/<id>/, and /projects/<id>/bid/ all map to views in the same views module. The simple structure is ideal for smaller projects because all routing logic is consolidated, reducing the amount of folder navigation and cross-app imports.

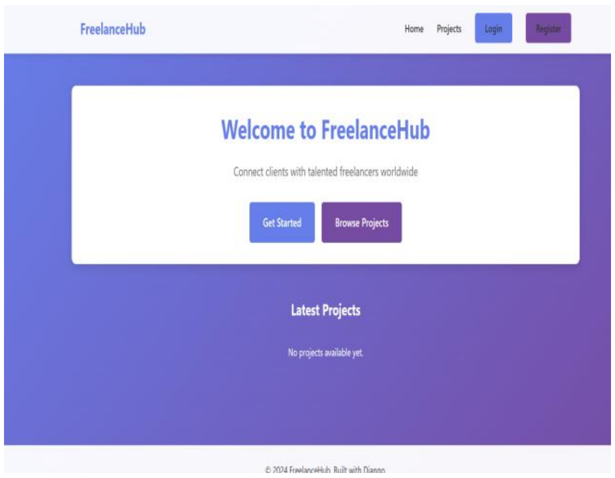
7.3 Important Functionalities

One of the main components of the system is **user authentication**, which allows users to register and log in. Using Django's built-in authentication, users create an account by entering their name, email, password, and selecting their role (client or freelancer). After successful registration, users can log in and are redirected to their dashboards, where the interface dynamically changes depending on their role. Clients see options such as "Post Project," whereas freelancers see "View Projects" to browse available work. Keeping user logic inside one app also makes role-based access easier because all role checks are done in the same views file.

Clients have the ability to **post projects** through a simple form. The project creation view validates the input, saves a new project model instance, and associates it with the logged-in client. Freelancers can **view all open projects** through a dedicated project listing page. When they open a specific project, they can see full details and also view previously placed bids. A freelancer can submit a **bid** by entering the amount and proposal message, which is stored in the Bid model. The system prevents clients from bidding and ensures only freelancers have access to the bid creation functionality. Meanwhile, clients can view all bids submitted for their project, compare freelancers, and make informed decisions.

Finally, the system enforces **role-based access and security** by verifying user roles before allowing operations such as posting projects or placing bids. Django’s @login_required decorator ensures that only authenticated users can perform important actions. Additionally, conditional checks like if user.role == 'client' or if user.role == 'freelancer' help prevent misuse and maintain workflow integrity. By developing everything within one app, these role-based checks are centralized, making the implementation easier to maintain and extend in the future.

8. Screenshots



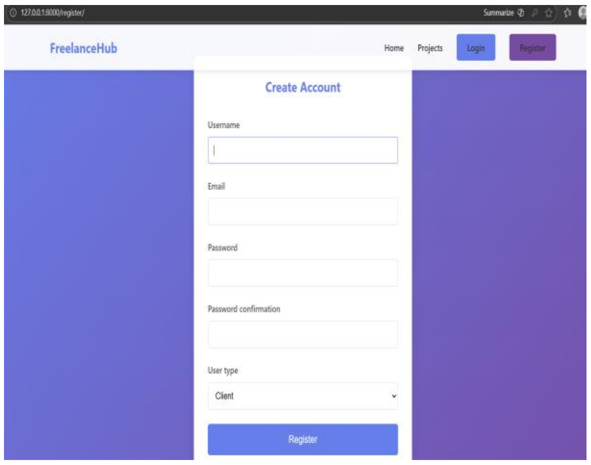
Welcome to FreelanceHub

Connect clients with talented freelancers worldwide

Latest Projects

No projects available yet.

© 2024 FreelanceHub. Built with Django



Create Account

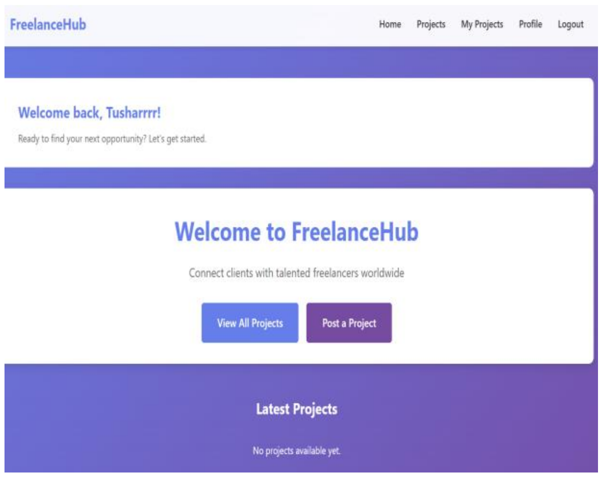
Username

Email

Password

Password confirmation

User type: Client



Welcome back, Tusharrrr!

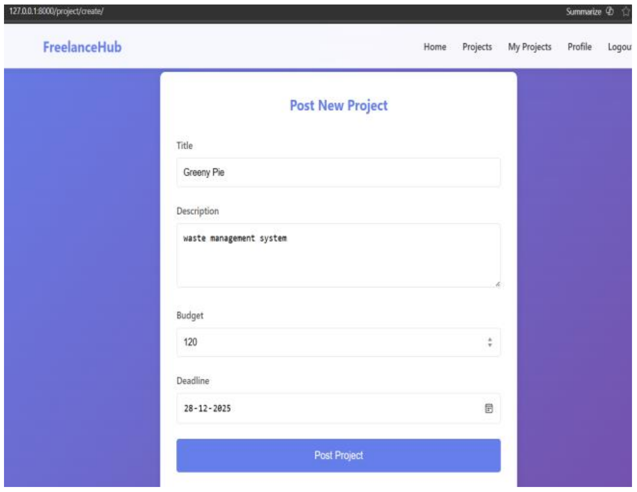
Ready to find your next opportunity? Let's get started.

Welcome to FreelanceHub

Connect clients with talented freelancers worldwide

Latest Projects

No projects available yet.



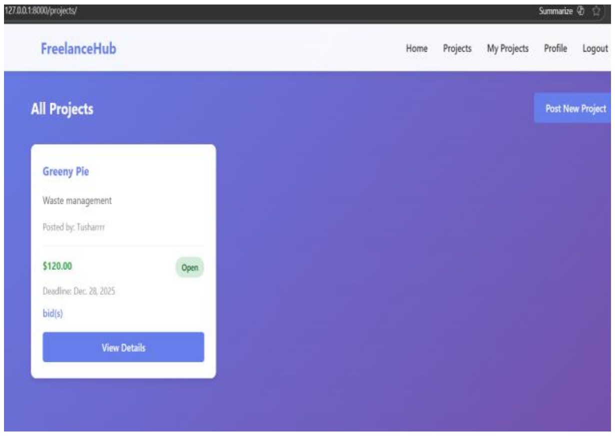
Post New Project

Title: Greeny Pie

Description: waste management system

Budget: 120

Deadline: 28-12-2025


127.0.0.1:8000/projects/

FreelanceHub — Home Projects My Projects Profile Logout

All Projects

Post New Project

Greeny Pie

Waste management

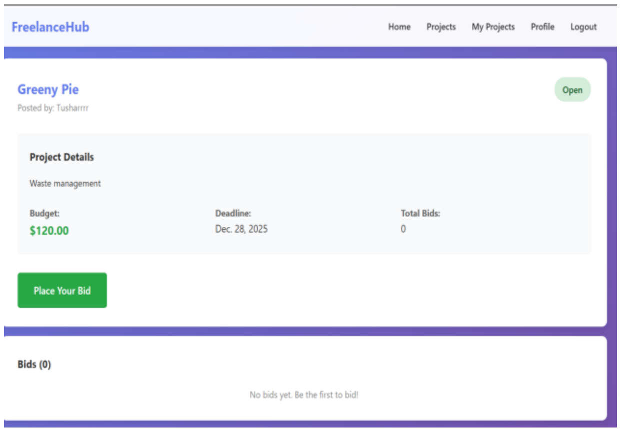
Posted by: Tusharrrr

$120.00 Open

Deadline: Dec. 28, 2025

bid(s)

View Details


FreelanceHub — Home Projects My Projects Profile Logout

Greeny Pie — Open

Posted by: Tusharrrr

Project Details

Waste management

Budget: $120.00

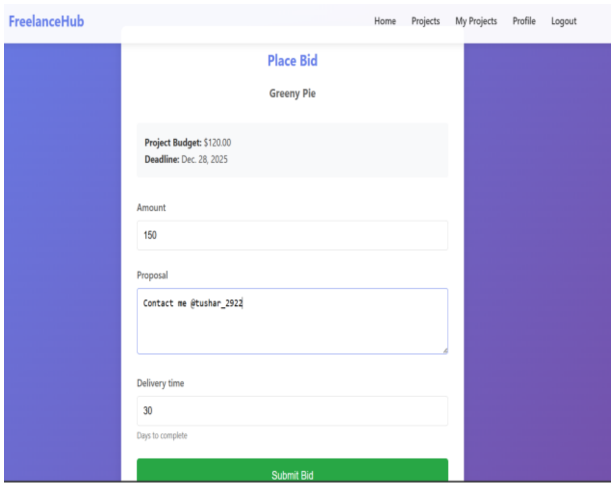
Deadline: Dec. 28, 2025

Total Bids: 0

Place Your Bid

Bids (0)

No bids yet. Be the first to bid!


FreelanceHub — Home Projects My Projects Profile Logout

Place Bid

Greeny Pie

Project Budget: $120.00
Deadline: Dec. 28, 2025

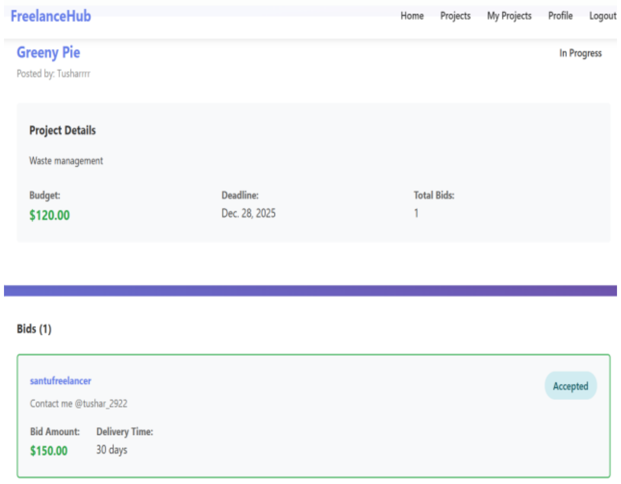
Amount: 150

Proposal: Contact me @tushar_2922

Delivery time: 30

Days to complete

Submit Bid


FreelanceHub — Home Projects My Projects Profile Logout

Greeny Pie — In Progress

Posted by: Tusharrrr

Project Details

Waste management

Budget: $120.00

Deadline: Dec. 28, 2025

Total Bids: 1

Bids (1)

santufreelancer — Accepted

Contact me @tushar_2922

Bid Amount: $150.00

Delivery Time: 30 days

9. Testing

Testing for the Mini Freelancing Platform focused on validating all key functionalities, including user authentication, project posting, bidding workflow, role-based restrictions, and data integrity. Since the platform uses Django's MVT architecture, both backend logic and frontend interactions were tested to ensure a smooth freelancing experience for clients and freelancers. Functional testing verified that each feature—such as posting a project, placing a bid, or viewing user dashboards—performed correctly across browsers and devices. Validation tests ensured that incomplete, incorrect, or malicious inputs were handled safely. Security testing concentrated on session handling, access control enforcement, CSRF protection, and preventing unauthorized actions like clients placing bids or freelancers posting projects. Additional edge-case testing included scenarios such as bidding on deleted projects, accessing restricted URLs, and handling duplicate bids. Admin panel operations, including user management and project monitoring, were also tested to confirm proper CRUD operations. Overall, the testing phase demonstrated that the system behaves reliably, protects data integrity, and maintains a consistent workflow across user roles.

9.1 Form Validation

All user-facing forms in the Mini Freelancing Platform—such as registration, login, project creation, and bid submission—underwent thorough client-side and server-side validation. For registration forms, required-field checks ensured that users could not submit empty fields for name, email, password, or role. Invalid email formats, weak passwords, and password mismatches triggered clear error messages. In project creation forms, missing fields like title or description returned validation errors, while illegal values such as negative budgets or excessively long text were rejected. Bid forms were tested by submitting blank messages, invalid numeric values for bid amounts, and multiple bids from the same freelancer to ensure the system responds correctly. Cross-field validation checks included confirming that only freelancers can access the bid form and only clients can post projects. Injection attempts—such as entering SQL-like payloads in text fields—were validated to ensure Django's ORM prevented malicious execution. Automated tests using Django's Test Client, combined with simple JavaScript checks, ensured that invalid submissions never reached the database and that users always received user-friendly error feedback.

9.2 Login (Authentication & Authorization)

Authentication tests verified the correct handling of all login flows. Successful login using valid credentials resulted in proper session creation and redirection to the correct dashboard based on user role. Invalid login attempts, such as incorrect passwords or unregistered emails, displayed non-descriptive error messages to avoid information leakage. Logout testing confirmed that sessions were destroyed correctly and that users were redirected to the login page. Authorization testing ensured that clients and freelancers only accessed features permitted by their roles: for example, only clients could access the “Post a Project” page, and only freelancers could access “Place a Bid” views. Attempts by anonymous users to visit protected URLs loaded appropriate redirect responses or 403 error pages. Tests also verified session cookie parameters such as HttpOnly and Secure in production-like environments. Additional edge-case tests included multiple browser tab sessions, verifying that session consistency was maintained across simultaneous interactions. These tests confirmed that authentication logic is secure, reliable, and role-appropriate.

9.3 CRUD Operations

CRUD testing covered all core models in the system: Users, Projects, and Bids. For the Project model, tests validated that clients could successfully create valid projects, view project listings, update project details, and delete their own projects when needed. Deletion rules were tested to ensure that removing a project also removed associated bids (via cascading behavior) and did not leave orphaned records. For Bids, test cases included creating new bids, preventing duplicate bids from the same freelancer on a single project, and retrieving bid lists correctly on the project detail page. Update tests confirmed that clients could edit project details while freelancers could update or withdraw bids if allowed by the system design. For User CRUD, the admin interface was used to validate operations such as editing user details and disabling accounts. Foreign-key integrity tests ensured that invalid references could not be created, and concurrency tests verified that two freelancers bidding at the same time did not corrupt data. Automated model tests and integration tests using Django's Test Client confirmed correctness in all CRUD operations.

9.4 Error Handling & Resilience

Error handling tests ensured that the platform displays proper error messages and maintains stability during unexpected conditions. All 4xx errors—such as missing pages, unauthorized access, invalid form submissions, and malformed requests—were validated to show user-friendly messages instead of raw system tracebacks. 5xx server error simulations, such as temporary database unavailability, confirmed that the platform fails gracefully and logs exceptions properly. The system was tested against partial or interrupted project submissions, network delays during form postings, and failed bid submissions; in each case, the system preserved data integrity and avoided inconsistencies. Access violations, such as freelancers attempting to post projects or clients attempting to place bids, were handled by clean redirects or permission-denied responses. Logging tests confirmed that errors were recorded in Django's logging system for debugging purposes. Basic load tests simulated multiple users viewing projects and placing bids simultaneously to ensure the platform remained stable under moderate traffic. These resilience checks verified that the Mini Freelancing Platform maintains reliability even under error conditions or edge-case scenarios.

10. Results

The Mini Freelancing Platform successfully integrates all core components required for a simplified online freelancing system into a single, streamlined application. One of the key outcomes of the project is its ability to create a functional bridge between clients seeking work to be done and freelancers offering their skills. By providing features such as user registration, role-based dashboards, project posting, and structured bidding mechanisms, the platform effectively simulates the essential workflow of real-world freelancing websites. Clients can easily post detailed project requirements, while freelancers can view available work and submit competitive bids. The system ensures that every interaction—whether posting, browsing, or bidding—is recorded accurately in the database, enabling traceability and offering a smooth user experience. This demonstrates that the platform fulfills its purpose of providing a lightweight and accessible freelancing environment, especially suitable for students, beginners, and small-scale service providers.

Another strong result of the project is the effectiveness of its **workflow clarity and data management**. The system reliably handles multiple users, projects, and bids without conflict, ensuring that each project displays the correct set of bids and each freelancer sees accurate bid histories. Role-based access control has been implemented successfully, preventing unauthorized actions and ensuring that clients and freelancers can only perform tasks relevant to their roles. The project structure ensures that no invalid operations occur—clients cannot bid on projects, freelancers cannot post projects, and anonymous users cannot access protected pages. The detail page for each project effectively brings together project information and associated bids, giving clients a clear overview of available proposals. By maintaining clean navigation and predictable behavior, the system demonstrates high usability and correctness in workflow execution.

A major result of the platform is its ability to **promote fair and transparent interactions** between clients and freelancers. The bidding system enables freelancers to submit individualized proposals with pricing and messages, while clients can compare multiple bids before choosing the best freelancer for their needs. This competitiveness mirrors the behavior of professional freelancing marketplaces and demonstrates that the core logic of matching supply and demand works effectively within the application. The system's database relationships, built using Django's ORM, ensure consistent linking between users, projects, and bids. This results in reliable data retrieval, accurate list views, and consistent detail pages across the platform. The clean and validated data flow confirms that the system's backend architecture is both stable and scalable.

11. Conclusion

The Mini Freelancing Platform successfully demonstrates how a simplified online marketplace can connect clients with freelancers through an intuitive and streamlined workflow. By integrating core functionalities such as user registration, role-based dashboards, project posting, and bid submission, the system replicates the essential behavior of real-world freelancing portals while maintaining simplicity suitable for academic or small-scale use. The project achieves its primary objective of creating a space where clients can easily share their work requirements and freelancers can respond with competitive proposals, enabling meaningful interactions and efficient matching of skill to opportunity.

The platform's implementation using Django ensures reliable data handling, secure authentication, and clean separation of roles. The use of a single consolidated app and Django's ORM made development efficient while maintaining clarity in the architecture. Throughout the system, relationships between users, projects, and bids are handled seamlessly, demonstrating the correct use of relational database principles. The system's ability to accurately record project information, manage multiple bids, and prevent unauthorized actions validates both the correctness and robustness of the underlying logic.

From a usability perspective, the platform provides a clean, easy-to-understand interface that supports smooth navigation for both clients and freelancers. Important freelancing actions—such as browsing projects, submitting bids, and reviewing offers—are implemented in a way that mirrors real marketplace behavior, making the platform both educational and practically useful. The project showcases how even a simplified freelancing environment can help students or beginners understand digital marketplace workflows and user interactions.